



香港中文大學
The Chinese University of Hong Kong

CENG3420

Lab Overview & Introduction to RARS

Fangzhou Liu

Department of Computer Science & Engineering
Chinese University of Hong Kong
{fzliu23}@cse.cuhk.edu.hk

Spring 2026

Outline

- ① Overview of CENG3420 Labs
- ② RISC-V ISA Simulator – RARS

Overview of CENG3420 Labs

Assembler & Simulator

- Assembly language – symbolic
- Machine language – binary
- **Assembler** is a program that
 - turns symbols into machine instructions (e.g., riscv64-unknown-elf-as)
- **Simulator** is a program that
 - mimics the behavior of a processor
 - usually written in high-level language (e.g., spike)

We have 3 labs in total, with 2-3 sub-labs for each lab.

- **Lab1:** RISC-V assembly language programming using **RARS** simulator.
- In lab1, we will practice coding in RISC-V assembly language, and understand how our codes run in a RISC-V CPU.
 - **Lab1-1:** basic operators and system call.
 - **Lab1-2:** function call and simple algorithm implementation.
 - **Lab1-3:** stack data structure, recursive function call, more complex algorithm implementation.

We have 3 labs in total, with 2-3 sub-labs for each lab.

- **Lab2:** build(complete) a C-based RISC-V assembler and simulator.
- Codebase: <https://github.com/seanzw/ CUHK-CENG3420-Lab>. We need to implement the assembler and simulator based on the codebase.
 - Lab2-1: implement a RISC-V assembler.
 - Lab2-2: implement a RISC-V ISA simulator with:
 - RISC-V 32 general-purpose registers
 - 32-bit data and address
 - 25+ instructions (including pseudo instructions)

We have 3 labs in total, with 2-3 sub-labs for each lab.

- **Lab3:** build a more complete C-based RISC-V Simulator based on lab2.
 - Lab3-1: control logic in CPU, finite state machine.
 - Lab3-2: execution model, memory interface.
 - Lab3-3: BUS driver, etc.

RISC-V ISA Simulator – RARS

What is RARS

- **RARS is the RISC-V Assembler, Runtime and Simulator for RISC-V assembly language programs.**
- We write codes in RISC-V assembly language, then **RARS** translates them into RISC-V instructions and corresponding machine codes, then execute the codes through simulation, like a RISC-V CPU.
- **RARS** supports RISC-V IMFDN ISA base (riscv32 & riscv64).
- **RARS** supports debugging using breakpoints like *ebreak*.
- **RARS** supports side by side comparison from psuedo-instruction to machine code with intermediate steps.

Launch RARS

- RARS tutorial: <https://cass-kul.github.io/tutorials/rars/>
- Install Java environment: <https://java.com/en/download/>
- Download RARS:
<https://github.com/TheThirdOne/rars/releases/tag/continuous>
- Run RARS: run command `java -jar <rars jar path>` in the command window, under the path where you place `rars.jar`

```
cbai@hpc1:/research/dept8/gds/cbai/ta/rars$ java -jar rars.jar
```

- We also provide Java install package and RARS in **RARS.zip** on **Blackboard**.

RARS Overview

The screenshot shows the RARS assembly editor interface. On the left, the assembly code for `hello3420.asm` is displayed:

```
.globl _start
.data # global variable declarations follow this line
welcome_msg: .asciz "Welcome to CENG3420!\n"
.text # instructions follow this line
_start: # a label, marks a position in the code
    addi $0, $0, 1 # STDOUT=1
    la $1, welcome_msg # Load the address of welcome_msg
    addi $2, $0, 21 # Length of the string
    addi $7, $0, 64 # Specify the system call number
    ecall # Raise a system call
# End of program, leave a blank line afterwards is preferred
```

The Registers panel on the right lists the following registers with their values:

Name	Number	Value
zero	0	0x00000000
r0	1	0x00000000
\$0	2	0xffffeffe
GP	3	0x10008000
TP	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
s0	10	0x00000000
s1	11	0x00000000
s2	12	0x00000000
s3	13	0x00000000
s4	14	0x00000000
s5	15	0x00000000
s6	16	0x00000000
s7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x04000000

The Messages panel at the bottom shows the output of the program:

```
Welcome to CENG3420!
— program is finished running (dropped off bottom) —
Clear
Reset: reset completed.
```

RARS edit panel

RARS Overview

The screenshot displays the RARS debugger interface with four main panels:

- Source codes panel**: Shows assembly code for a file named `test.asm`. The code includes directives like `# 12 "size/rv64ui/swrlw.s"`, `.text`, `.globl _start`, `_start: nop`, and various arithmetic tests involving `li`, `srliw`, and `bne` instructions.
- Registers panel**: A table showing the state of 32 registers (r0 to r31) and floating-point registers (x0 to x7). All registers contain zero.
- Program information panel**: Displays assembly output from the `assemble` command, showing the assembly source and the resulting object code.

RARS Overview

The screenshot shows the RARS execute panel interface. At the top is a menu bar with File, Edit, Run, Settings, Tools, Help, and a toolbar with various icons. A status bar at the top right says "Run speed at max (no interaction)".

The main area contains several panes:

- Registers:** Shows a table of registers with their names, numbers, and values.
- Floating Point:** Not visible in the screenshot.
- Control and Status:** Not visible in the screenshot.
- Text Segment:** An assembly code editor with columns for Bkpt, Address, Code, Basic, and Source. The source column contains comments like "# STDOUT=1" and "# Load the address of welcome_msg".
- Data Segment:** A table showing memory values at addresses from 0x10010000 to 0x10010100.
- Messages:** A pane showing the output of the program, including "Welcome to CERN3420!" and "program is finished running (dropped off bottom)". It includes a "Clear" button.
- Run I/O:** A pane below the Messages pane showing "Reset: reset completed."

RARS execute panel

RARS Overview

The screenshot shows the RARS debugger interface with several panels:

- Tools panel**: Located at the top right, showing assembly code in the Text segment panel.
- Text segment panel**: Shows assembly code for the .text section. The current instruction is highlighted in yellow: `99 test_2 li xl, 0xffffffffffff000000`. Other instructions include `90 _start: .text`, `100: mulr xl xl, 0`, `101: li xl, 0xffffffffffff000000`, `102: li sp, 2`, `103: bne xl xl, fail`, and `105: test_3 li xl, 0xffffffffffff000000`.
- Data segment panel**: Shows memory dump data from address 0x10010000 to 0x10010140. All values are 0x00000000.
- Registers panel**: Shows the state of 32 general-purpose registers (r0 to r31) and floating-point registers (f0 to f7). All register values are 0x00000000.
- Program information panel**: Displays assembly output for the file F:\Research\mine\TA\CE803420\test\test.asm. It shows errors: "Warning in F:\Research\mine\TA\CE803420\test\test.asm line 312 column 2: RARS does not recognize the _global directive. Ignored." and "Warning in F:\Research\mine\TA\CE803420\test\test.asm line 318 column 2: RARS does not recognize the _global directive. Ignored." The message "Assemble: operation completed successfully." is also present.

Shortcuts in Windows

- Create a new source file: Ctrl + N
- Close the current source file: Ctrl + W
- Assemble the source code: F3
- Execute the current source code: F5
- Step running: F7
- Instructions & System call query: F1

An Example Program

Hello CENG3420

```
.globl _start

.data    # global variable declarations follow this line
welcome_msg: .asciz "Welcome_to_CENG3420!\n"

.text    # instructions follow this line
_start: # a label, marks a position in the code
        addi a0, x0, 1 # STDOUT=1
        la a1, welcome_msg # Load the address of welcome_msg
        addi a2, x0, 21 # Length of the string
        addi a7, x0, 64 # Specify the system call number
        ecall # Raise a system call
# End of program, leave a blank line afterwards is preferred
```

An Example Program

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x00100513	addi x10,x0,1	9: addi a0, x0, 1
	0x00400004	0x0fc10597	auipc x11,0x0000fc10	11: la s1, welcome_msg
	0x00400008	0xffffc58593	addi x11,x11,0xffff...	
	0x0040000c	0x01500613	addi x12,x0,21	13: addi a2, x0, 21
	0x00400010	0x04000093	addi x17,x0,0x00000040	15: addi a7, x0, 64
	0x00400014	0x00000073	ecall	17: ecall

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x00400000	0x00100513	0x0fc10597	0xffffc58593	0x01500613	0x04000093	0x00000073	0x00000000	0x00000000
0x00400020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x004000a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x004000c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x004000e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x00400100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x00400000 {.text} Hexadecimal Addresses Hexadecimal Values ASCII

Messages Run I/O

Welcome to ENG3420!

-- program is finished running (dropped off bottom) --

Control and Status Registers Floating Point

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffefffc
gp	3	0x10000000
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000015
a1	11	0x10010000
a2	12	0x00000015
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000040
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x0040001c

System Calls in RARS

RARS provides a small set of operating system-like services through the system call (`ecall`) instruction. Register contents are not affected by a system call, except for result registers in some instructions.

- Load the service number (or number) in register a7.
- Load argument values, if any, in a0, a1, a2 ..., as specified.
- Issue `ecall` instruction.
- Retrieve return values, if any, from result registers as specified.

System Calls in RARS

Name	Number	Description	Inputs	Outputs
PrintInt	1	Prints an integer	a0 = integer to print	N/A
PrintFloat	2	Prints a float point number	fa0 = float to print	N/A
PrintString	4	Prints a null-terminated string to the console	a0 = the address of the string	N/A
ReadInt	5	Reads an int from input console	a0 = the int	N/A
ReadFloat	6	Reads a float from input console	fa0 = the float	N/A
ReadString	8	Reads a string from the console	a0 = address of input buffer, a1 = maximum number of characters to read	N/A
Open	1024	Opens a file from a path Only supported flags (a1), read-only (0), write-only (1) and write-append (9)	a0 = Null terminated string for the path, a1 = flags	a0 = the file descriptor or -1 if an error occurred
Read	63	Read from a file descriptor into a buffer	a0 = the file descriptor, a1 = address of the buffer, a2 = maximum length to read	a0 = the length read or -1 if error
Write	64	Write to a filedescriptor from a buffer	a0 = the file descriptor, a1 = the buffer address, a2 = the length to write	a0 = the number of characters written
LSeek	62	Seek to a position in a file	a0 = the file descriptor, a1 = the offset for the base, a2 is the begining of the file (0), the current position (1), or the end of the file (2)}	a0 = the selected position from the beginning of the file or -1 is an error occurred

THANK YOU!